저작자표시-비영리-변경금지 2.0 대한민국

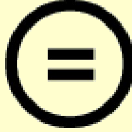
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.


변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

[UCI]I804:31001-200000225312

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

Master's Thesis

Development of Three-Dimensional Parallel Code to Study the Motions of Particles in a Fluid Using Lattice Boltzmann Method

Jeong-Gi Park

Department of Mechanical Engineering

Graduate School of UNIST

2019

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

Development of Three-Dimensional Parallel Code to Study the Motions of Particles in a Fluid Using Lattice Boltzmann Method

Jeong-Gi Park

Department of Mechanical Engineering

Graduate School of UNIST

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

Development of Three-Dimensional Parallel Code to Study the Motions of Particles in a Fluid Using Lattice Boltzmann Method

A thesis/dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Jeong-Gi Park

Month/Day/Year of submission
Approved by

____

Advisor
Chun Sang Yoo

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Development of Three-Dimensional Parallel Code to Study the Motions of Particles in a Fluid Using Lattice Boltzmann Method

Jeong-Gi Park

This certifies that the thesis/dissertation of Jeong-Gi Park is approved.

June/14/2019

signature

Advisor: Chun Sang Yoo

signature

Hyungson Ki

signature

Jaesung Jang

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Abstract

The three-dimensional parallel code is developed for the lattice Boltzmann method. It is to simulate multiphase flows containing particles. The code is the combination of the two models, the Shan-Chan multiphase model for a viscous fluid, the pseudo-solid model for particles. The difficulties in implementing the methods and some possible optimization techniques are suggested. This code can be used to simulate the dynamics of the self-assembly driven by evaporation and any multiphase flow with different sizes of particles.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Contents

1. Introduction---------- 9

2. Simulation methods---------- 11

2.1 Lattice Boltzmann Equation ---------- 11

2.2 Shan-Chan Two Component Model ---------- 14

2.3 Particle Model ---------- 16

2.3.1 Particle domain ---------- 16

2.3.2 Particle-fluid interaction ---------- 18

2.3.3 Particle-particle interaction ---------- 18

3. Details of the Program ---------- 20

3.1 Velocity Set---------- 21

3.2 Domain Decomposition---------- 22

3.3 `FLUID` Datatype ---------- 22

3.4 Neighboring Processors---------- 23

3.5 Sequence of Substeps ---------- 24

3.6 Initialization---------- 26

3.7 Boundary Conditions---------- 28

3.8 Communications ---------- 28

3.9 Two Component Model---------- 31

3.10 Particle Model---------- 32

3.10.1 Density---------- 32

3.10.2 Local velocity ---------- 36

3.10.3 Force gathering---------- 37

3.10.4 Position update---------- 39

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

4. Examples ---------- 41

5. Conclusions ---------- 43

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

1. Introduction

Matter consists of molecules. It is now an obvious statement, but things are somewhat different at Ludwig Boltzmann's era. Even though Boltzmann established a foundation for kinetic theory and the statistical physics based on the concepts of molecules and atoms, the mainstream of respected physicist did not accept the atomic theory. The attack to his theory was severe enough for him to be suffered from depression. Now the atomic theory is a foundation of modern theory and provide a rigorous foundation of the simulation method such as the molecular dynamics and the lattice Boltzmann method (LBM).

The lattice Boltzmann equation can be derived from the Boltzmann transport equation [1], which describes the behavior of gas based on the molecular nature of fluid. It has been emerging as one of the promising computational fluid dynamic (CFD) method in the last few decades because of its simplicity, ability to handle the complex geometry and physics, and applicability on multicore computing systems. Although LBM is currently considered as a discretized numerical scheme for the Boltzmann transport equation [2], but it is worth to mention that it was originally from the lattice gas automata (LGA) [3-5], which stems from the cellular automata (CA) [6]. Knowledge on these subject is not essential to study the LBM, but a brief glance at Wolf-Gladrow's book [7], which describes connection between the LGA and the LBM concisely, will be useful because some early papers on the LBM are not easy to follow without being familiar with the CA and the LGA.

The continuum assumption is the most fundamental assumption of the conventional fluid mechanics. It enables us to describe the motion of fluid with a differential equation, the Navier-Stokes equation. Various discretization methods have been developed to efficiently solve it. It is inherently nonlocal because it deals with the gradients of quantities, which require information from adjacent nodes. On the other hand, the molecular dynamics is a microscopic description of the fluid behavior, so Newtonian dynamics can be used to describe the motion of each particle and its computation is local. However, the intermolecular forces need to be determined with the quantum mechanics, and it is impossible to track every single molecule in human-scale systems. The LBM is somewhere between them, and it is called mesoscopic description of fluid. Its fundamental quantity is the particle distribution functions, or the representative collection of molecules [8].

The LBM's kinetic nature enables us to simulate multiphase fluid in a relatively straightforward way. Using conventional CFD approaches, the sharp gradients of fluid properties in the interface requires additional modeling. Furthermore, the tracking of phase interfaces and phase change are not simple tasks. Whereas, from a microscopic view, a multiphase flow is a simple phenomenon resulted from molecular interactions. Three most popular models in LBM are the color-gradient model [9], the Shan-Chan model [10-14], and the free-energy model [15-16]. After they are first introduced, they have been

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

actively studied and improved. The recent improvement and theoretical basis of the three models are well organized in Huang's recent work [17].

Another the strength of LBM is its ability to handle particulate flow. Computation in LBM is local, so the LBM model can effectively calculate the interaction between fluid and particles. After Ladd [18-22] proposed a particle model, LBM has been widely used to study various phenomena such as emulsions [23], coffee ring effect [24], blood flow [25], and capillary force [26]. Meanwhile, Liang et al. [27] suggested a novel method called pseudo-solid model (PSM) to model solid particle without bounce-back moving boundary condition. In this paper, the features of PSM is analyzed and discussed.

Above all the advantage of the LBM, the scalability on parallel computers, is arguably the primary reason for the increasing interest of the LBM. Performance of a single microprocessor increased about 50% annually from 1986 to 2002 [28]. However, the increasing rate has reduced to about 20% after 2002 [29]. Therefore, high performance can be only achieved with multiple processors to successfully simulate the modern scientific and engineering problems such as combustion at high pressure, the weather forecast, and the turbulent flow.

Now the LBM has broadened its area to commercial software products. DASSAULT SYSTEMS takes over PowerFLOW, which was the first commercial software based on LBM, in 2017. NUMECA in 2018 is merged with Palabos, which was the most popular and versatile open source software in LBM community. Although developing in-house code in a lab may look like an impractical task when there already exist reliable and powerful software, the LBM is still new methods that require elaborate improvements. Indeed a variety of new models specific to problems are suggested every. In-house codes will help one who wants to develop or adopt them for their own problems. Additionally, a simple in-house code is easy to read without profound deep knowledge on programming technique, and useful to getting basic understanding about the method.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

2. Simulation Method

Lattice Boltzmann method is a relatively young branch of CFD, so different books use different notations and take different approaches. This has been one of the obstacles to beginners of the LBM. Fortunately, recently Kruger et al. (2017) [8] published a comprehensive, well organized textbook, and this paper follows the notations and the approach from it. They explain the LBM as a discretization scheme of the Boltzmann transport equation, and I briefly introduced the result here. Readers who are interested in how the LBM is originally grew out of CA and LGCA can refer Wolf-Gladrow (2006) [7] and Sukop and Thorne (2006) [30].

2.1 Lattice Boltzmann Equation.

The Boltzmann transport equation describes behavior of gas using a mesoscopic variable, distribution function $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$. It can be considered as generalized density whose value depends on not only space $\boldsymbol{x}$ but also microscopic particle velocity space $\boldsymbol{\xi}$ [8]. Mass density ρ can be obtained by integrating distribution function over velocity space as, $\rho(\boldsymbol{x}, t) = \int f(\boldsymbol{x}, \boldsymbol{\xi}, t)\, d^3\xi$. If we take the derivative of the distribution function with respect to time *t* using the chain rule,

$$\frac{d}{dt} f(\boldsymbol{x}(t), \boldsymbol{\xi}(t), t) = \frac{\partial f}{\partial t}\frac{dt}{dt} + \frac{\partial f}{\partial x_\beta}\frac{dx_\beta}{dt} + \frac{\partial f}{\partial \xi_\beta}\frac{d\xi_\beta}{dt} \tag{2.1}$$

where the index notation was used to denote each component of the position and velocity vector. The total derivative $\mathrm{d}f/dt$ is conventionally written as $\Omega(f)$. Using $\mathrm{d}x_\beta/dt = \xi_\beta$ and $\mathrm{d}\xi_\beta/dt = F_\beta/\rho$, where F_β is the specific body force, we have the Boltzmann transport equation

$$\frac{\partial f}{\partial t} + \xi_\beta \frac{\partial f}{\partial x_\beta} + \frac{F_\beta}{\rho}\frac{\partial f}{\partial \xi_\beta} = \Omega(f). \tag{2.2}$$

The left-hand side is a linear partial differential equation, because the non-linear nature of fluid is inherent in $\Omega(f)$ on the right-hand side. It is called the collision integral and describe the effect of collisions of particles on the distribution function. Its original form is involved integral equation, but Bhatnagar, Gross, Krook suggested much simpler form [31]

$$\Omega(f) = -\frac{1}{\tau}(f - f^{eq}) \tag{2.3}$$

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

where τ is called relaxation time or collision time, related to time between the collisions. This simplification may look immoderate considering complexity of original integral equation, but the fact that the detailed process of collision of molecules hardly have effects on the macroscopic properties of fluid makes it a reasonable assumption. A tremendous number of microstates correspond to one macrostate, so we can simply choose most manageable one. Additionally, the collisions tend to diffuse the momentums of each particle, and after a sufficient number of collisions occurs, system will settle down in an equilibrium state. Therefore, the time rate of change of distribution function is proportional to how fall it deviates from the equilibrium and inversely proportional time between the collisions. The equilibrium distribution was calculated by Maxwell and Boltzmann as

$$f^{eq}(\mathbf{x}, |\boldsymbol{v}|, t) = \rho \left(\frac{1}{2\pi RT}\right)^{3/2} e^{-|v|^2/(2RT)} \tag{2.4}$$

where $\boldsymbol{v}(\boldsymbol{x}, t)$ is the difference between the microscopic particle velocity and the macroscopic velocity of fluid.

The actual discretization process requires some mathematical background on Hermite series expansion and is beyond the scope of this paper. The mathematically rigorous procedure is explained in detail on Shan et al. (2006) [2]. After discretizing and normalizing process [7], the Boltzmann transport equation without force term becomes

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = -\frac{1}{\tau/\Delta t}\left(f_i(\boldsymbol{x}, t) - f_i^{eq}(\boldsymbol{x}, t)\right) \tag{2.5}$$

which is the lattice Boltzmann equation. Here f_i means the distribution functions corresponding to the discretized finite velocity set, $\boldsymbol{c}_i$. The normalized lattice spacing Δx and time step Δt are commonly set to both 1 lattice unit to simplicity. The LBM simulation and real system can be compared using dimensionless numbers such as Reynolds number, Bond number and Capillary number. In every simulations of this paper, $\tau/\Delta t$ is set to 1 for simplicity. Hence, τ is equal to Δt, which means the system is in equilibrium state at each time step because relaxation time is equal to time step. Then equation (2.5) becomes

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) = f_i^{eq}(\boldsymbol{x}, t).$$

There are a couple of the possible velocity set with the weight coefficient w_i. They are classified using notation DdQq, where d is the number of dimensions and q is the number of discretized velocities. D3Q19 is used in this paper and the numbering is shown in below figure.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

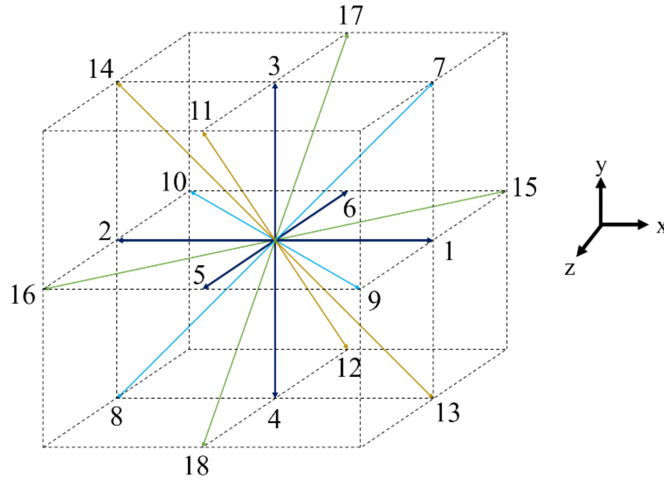


Figure 1. D3Q19 velocity set

The discretized equilibrium distribution function is given by

$$f_i^{eq}(\boldsymbol{x},t) = w_i\rho\left[1+\frac{\mathbf{u}\cdot\mathbf{c}_i}{c_s^2}+\frac{(\mathbf{u}\cdot\mathbf{c}_i)^2}{2c_s^4}-\frac{\mathbf{u}\cdot\mathbf{u}}{2c_s^2}\right] \tag{2.6}$$

where c_s is a constant determined by the velocity set used. For every velocity set used in this paper c_s has a value of 1/3. Its theoretical meaning can be found in discretization procedure [2].

For implementation, it is convenient rewrite lattice Boltzmann equation as

$$f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) = f_i(\boldsymbol{x},t)\left(1-\frac{1}{\tau/\Delta t}\right)+f_i^{eq}(\boldsymbol{x},t)\frac{1}{\tau/\Delta t}. \tag{2.7}$$

Calculating the right-hand side is often called the collision step and denoted with $f_i^*(\boldsymbol{x},t)$. The actual calculation step can be understood easily with the simplest velocity set, D1Q3. At some time t and at some point x,

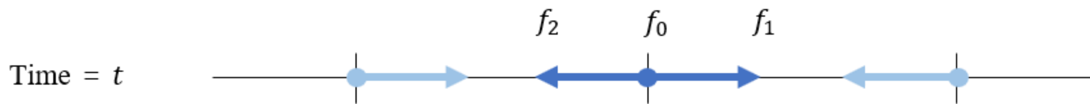


Figure 2. Distribution function at time *t*.

We can find density ρ and momentum density $\rho\boldsymbol{u}$ as

$$\rho(\boldsymbol{x},t) = \sum_i f_i(\boldsymbol{x},t), \qquad \rho\boldsymbol{u}(\boldsymbol{x},t) = \sum_i \boldsymbol{c}_i f_i(\boldsymbol{x},t). \tag{2.8}$$

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Then the equilibrium distribution can be found with the density and the velocity. After evaluating the collision step,

$$f_i^*(\boldsymbol{x},t) = f_i(\boldsymbol{x},t)\left(1-\frac{1}{\tau/\Delta t}\right)+f_i^{eq}(\boldsymbol{x},t)\frac{1}{\tau/\Delta t} \quad (2.9)$$

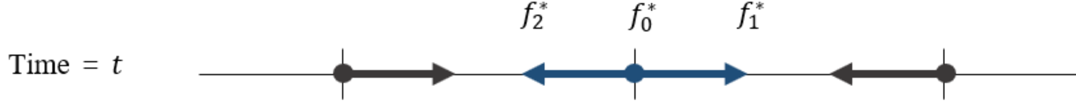


Figure 3. Distribution function after collision at time *t*.

Next step is called the streaming.

$$f_i(\boldsymbol{x}+\boldsymbol{c}_i\Delta t, t+\Delta t) = f_i^*(\boldsymbol{x},t) \quad (2.10)$$

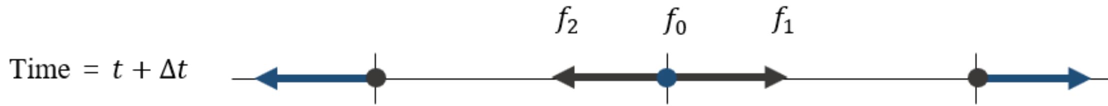


Figure 4. Distribution function after streaming.

Now ρ and $\rho\boldsymbol{u}$ at time $t+\Delta t$ can be computed.

2.2 Shan-Chan Two Component Model

The ideal gas equation of state is monotonic function, so no phase separation is expected theoretically. Whereas the van der Waals equation of state can explain the phase separation by taking account of the intermolecular forces and volumes of molecules. Based on this idea, Shan and Chen [10] proposed one of the most popular multiphase model in lattice Boltzmann method. In this paper, only two component model will be introduced, but extending to arbitrary number of components is straightforward.

In the two component Shan-Chan model (SC model), two layers of domain is implemented with two distinct distribution functions, each denoted as red f_i^R and blue f_i^B

$$f_i^\sigma(\boldsymbol{x}+\mathbf{c}_i\Delta t, t+\Delta t) - f_i^\sigma(\boldsymbol{x},t) = -\frac{1}{\frac{\tau^\sigma}{\Delta t}}\left(f_i^\sigma(\boldsymbol{x},t) - f_i^{\sigma,eq}(\boldsymbol{x},t)\right), \quad (2.11)$$

$$f_i^{\sigma,eq}(\boldsymbol{x},t) = w_i\rho^\sigma\left[1+\frac{\mathbf{u}^{\sigma,eq}\cdot\mathbf{c}_i}{c_s^2}+\frac{(\mathbf{u}^{\sigma,eq}\cdot\mathbf{c}_i)^2}{2c_s^4}-\frac{\mathbf{u}^{\sigma,eq}\cdot\mathbf{u}^{\sigma,eq}}{2c_s^2}\right], \quad (2.12)$$

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

where σ denotes either R or B. The density and momentum density is given by $\rho^\sigma(\boldsymbol{x},t) = \sum_i f_i^\sigma(\boldsymbol{x},t)$ and $\rho^\sigma \boldsymbol{u}^\sigma(\boldsymbol{x},t) = \sum_i \boldsymbol{c_i} f_i^\sigma(\boldsymbol{x},t)$. Without interaction force between two fluid components, this system can be regarded as an ideal gas mixture and have some the common equilibrium velocity, $\mathbf{u}^{eq}$[11]. Momentum conservation can be used to find appropriate expression for $\mathbf{u}^{eq}$. After multiplying $\boldsymbol{c}_i$ to both side of above equation, and summing over the index *i*,

$$\sum_i [f_i^\sigma \boldsymbol{c}_i(\boldsymbol{x}+\boldsymbol{c}_i, t+1) - f_i^\sigma \boldsymbol{c}_i(\mathbf{x},t)] = \sum_i \left[-\frac{f_i^\sigma \boldsymbol{c}_i(\mathbf{x},t) - f_i^{\sigma,eq}\boldsymbol{c}_i(\mathbf{x},t)}{\tau^\sigma/\Delta t}\right] \tag{2.13}$$

which can be rewrite with definition of the momentum density as

$$\rho^\sigma \mathbf{u}^\sigma(\mathbf{x}+\boldsymbol{e}_i, t+1) - \rho^\sigma \mathbf{u}^\sigma(\mathbf{x},t) = -\frac{\rho^\sigma \mathbf{u}^\sigma(\mathbf{x},t) - \rho^\sigma \mathbf{u}'(\mathbf{x},t)}{\tau^\sigma/\Delta t} = \Delta_c \mathbf{P}^\sigma(\mathbf{x},t) \tag{2.14}$$

where $\Delta_c \mathbf{P}^\sigma(\mathbf{x},t)$ is momentum change of fluid component by collision process. Since the collision step conserves total momentum [11],

$$\begin{aligned}\sum_\sigma \Delta_c \mathbf{P}^\sigma(\mathbf{x},t) &= \Delta_c \mathbf{P}^R(\mathbf{x},t) + \Delta_c \mathbf{P}^B(\mathbf{x},t) \\ &= -\frac{\rho^R \mathbf{u}^R(\mathbf{x},t) - \rho^R \mathbf{u}'(\mathbf{x},t)}{\frac{\tau^R}{\Delta t}} - \frac{\rho^B \mathbf{u}^B(\mathbf{x},t) - \rho^B \mathbf{u}^{eq}(\mathbf{x},t)}{\frac{\tau^B}{\Delta t}} = 0\end{aligned} \tag{2.15}$$

Solve for $\mathbf{u}_{ideal}^{eq}(\mathbf{x},t)$ to get

$$\mathbf{u}_{ideal}^{eq}(\mathbf{x},t) = \frac{\dfrac{\rho^R \mathbf{u}^R(\mathbf{x},t)}{\tau^R/\Delta t} + \dfrac{\rho^B \mathbf{u}^B(\mathbf{x},t)}{\tau^B/\Delta t}}{\dfrac{\rho^R}{\tau^R/\Delta t} + \dfrac{\rho^B}{\tau^B/\Delta t}}. \tag{2.16}$$

If we set $\tau^R/\Delta t = \tau^B/\Delta t = 1$, we have simpler form of

$$\mathbf{u}_{ideal}^{eq}(\mathbf{x},t) = \frac{\rho^R \mathbf{u}^R(\mathbf{x},t) + \rho^B \mathbf{u}^B(\mathbf{x},t)}{\rho^R + \rho^B}. \tag{2.17}$$

In order to simulate two immiscible fluid, Shan and Doolen introduced an interparticle force [11],

$$\boldsymbol{F}^{RB}(\boldsymbol{x}) = -\psi^R(\boldsymbol{x}) G^{RB} \sum_i w_i \psi^B(\boldsymbol{x}+\boldsymbol{c_i})\boldsymbol{c_i} \tag{2.18}$$

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

where $\boldsymbol{F}^{RB}$ means interparticle force on fluid red from fluid blue and $\boldsymbol{F}^{BR}$ can be found in the same way. $\psi^{\sigma}=\psi^{\sigma}(\rho^{\sigma})$ is called effective mass whose specific form determines the equation of state, and G is a constant to adjust the strength of the forces. G^{RB} should be equal to G^{BR} to ensure physical behavior [9]. It also determines the density ratio and surface tension [16]. In this paper, ψ^{σ} is always set as $\psi^{\sigma}=\rho^{\sigma}$.

To incorporate this force into lattice Boltzmann equation, several forcing schemes have been proposed []. Here the SC forcing scheme [10] is implemented as

$$\rho^{\sigma}\mathbf{u}^{\sigma,eq}(\mathbf{x},t)=\rho^{\sigma}\mathbf{u}_{\mathrm{ideal}}^{eq}+\tau^{\sigma}\mathbf{F}^{\sigma}. \qquad (2.19)$$

This $\mathbf{u}^{eq}$ will be used in collision step.

2.3 Particle Model

In the pseudo-solid model (PSM) [27], the force on a solid particle is calculated using Shan-Chan interaction force. In other words, the particles act like one of the Shan-Chan fluid components. The computation of fluid component is the same as SC model. Here the PSM is briefly summarized.

2.3.1 Particle domain

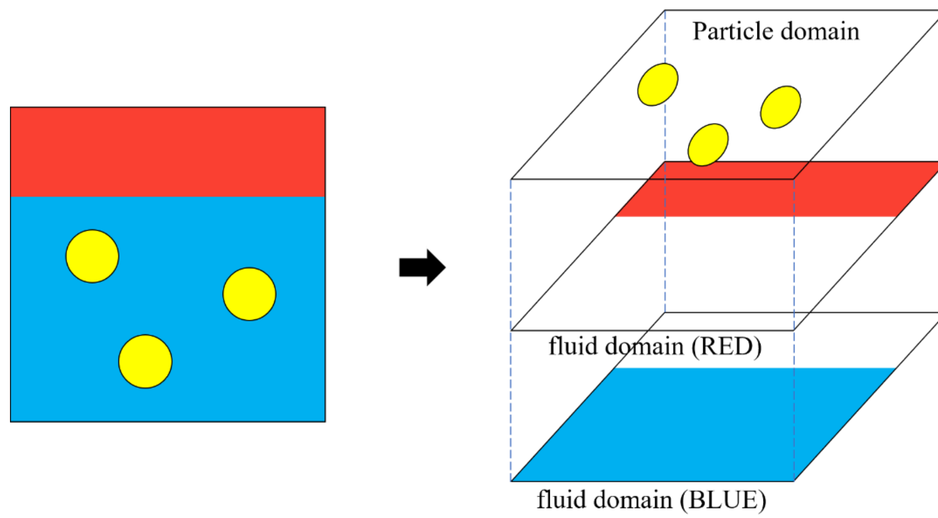


Figure 5. Three layers of domain for the PSM.

The PSM add one more layer of domain for particles to the double layer of fluid domain in two component SC model. The density of undeformable rigid particles whose radius is R is given by

$$n^P(\mathbf{x}) = \begin{cases} 1.0 & (r < R - 0.5\xi) \\ 0.0 & (r > R + 0.5\xi) \\ 1.0 + \dfrac{(R - 0.5\xi) - r}{\xi} & (R - 0.5\xi \le r \le R + 0.5\xi) \end{cases} \tag{2.20}$$

where ξ is a parameter for the interface width, which will be set to 1.

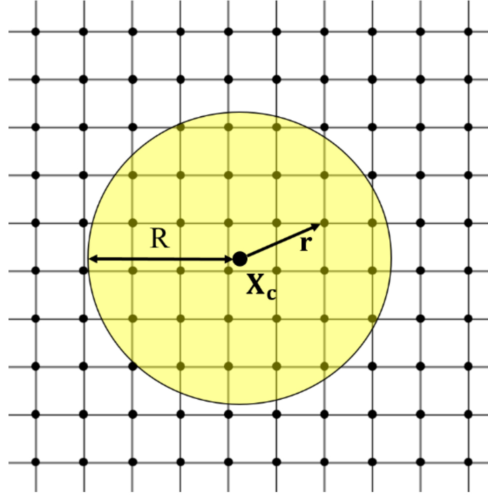


Figure 6. Particle density on the local domain.

The center of particle $\mathbf{X}_c$, translational velocity $\mathbf{V}_T$, angular velocity $\mathbf{\Omega}$ is updated by Newton's Law.

$$\frac{d\mathbf{X}_c}{dt} = \mathbf{V}_T, \tag{2.21}$$

$$M\frac{d\mathbf{V}_T}{dt} = \mathbf{F} = \mathbf{F}^G + \mathbf{F}^H, \tag{2.22}$$

$$I\frac{d\mathbf{\Omega}}{dt} = \mathbf{T} = \mathbf{T}^H, \tag{2.23}$$

where $\mathbf{F}^G$ means gravitational force on a particle and $\mathbf{F}^H$ is a hydrodynamic force from fluid on a solid particle.

2.3.2 Particle-fluid interaction

To compute the interaction between solid particle and fluid components, we consider the solid particle as a third fluid component – a particle component, which will be denoted with superscript P. At each grid point in a solid particle, local velocity of a particle component is determined by the particle's translational velocity and angular velocity.

$$\mathbf{V}^P(\mathbf{x}) = \mathbf{V}_T + \mathbf{V}_R = \mathbf{V}_T + \mathbf{\Omega} \times \mathbf{r}. \tag{2.24}$$

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

The common velocity of SC model at each point can be found by taking account the effect of the particle component.

$$\mathbf{u}_{ideal}^{eq}(\mathbf{x}) = \frac{\rho^P \mathbf{V}^P + \rho^R \mathbf{u}^R + \rho^B \mathbf{u}^B}{\rho^P + \rho^R + \rho^B}. \tag{2.25}$$

Note that where particle's density is zero, $\mathbf{u}_{ideal}^{eq}(\mathbf{x})$ will contain only terms for the red and blue fluid. The force on a fluid component now include the force from particle.

$$\boldsymbol{F}_{total}^{R}(\boldsymbol{x}) = -\psi^R(\boldsymbol{x})G^{RB}\sum_i w_i \psi^B(\boldsymbol{x}+\boldsymbol{c}_i)\boldsymbol{c}_i - \psi^R(\boldsymbol{x})G^{RS}\sum_i w_i \psi^P(\boldsymbol{x}+\boldsymbol{c}_i)\boldsymbol{c}_i, \tag{2.26}$$

where $\psi^P(\mathbf{x}) = n^P(\mathbf{x})$. Similarly, the second term vanishes where the particle's density is zero.

Hydrodynamic force on the particle by each component of fluid is determined by calculating momentum change of fluid at each node. By the momentum conservation, momentum change of a particle is the negative of the sum of momentum change of two fluid. Hence,

$$\Delta \boldsymbol{P}^S(\boldsymbol{x},t) = -\Delta P^F(\boldsymbol{x},t) = -\sum_\sigma \left[\rho^\sigma \boldsymbol{u}^\sigma(\boldsymbol{x},t) - \rho^\sigma \boldsymbol{u}_{ideal}^{eq}(\boldsymbol{x},t)\right] \tag{2.27}$$

The total hydrodynamic force on the solid particle can be obtain by summing over the particle's domain, the lattice grid points where $n^P(\mathbf{x}) \neq 0$

$$\mathbf{F}^H(t) = \sum_{\mathbf{x}} \Delta P^S(\mathbf{x},t), \tag{2.28}$$

$$\mathbf{T}^H(t) = \sum_{\mathbf{x}} \mathbf{r} \times \Delta P^S(\mathbf{x},t). \tag{2.29}$$

2.3.3 Particle-particle interaction

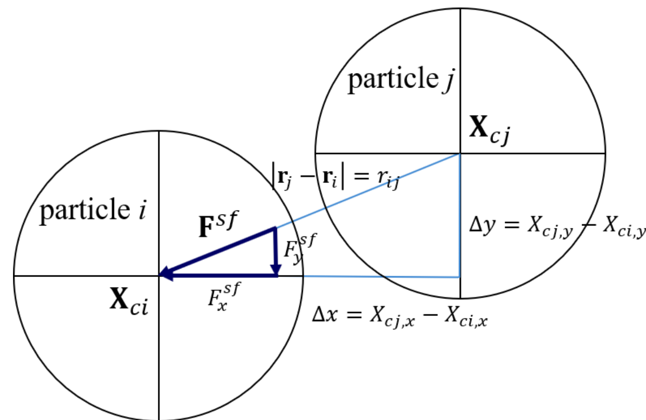


Figure 7. Relative position vector and force on a particle by the other particle.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

To prevent particle overlapping, a repulsive spring force [21-22] is implemented. Its magnitude is given by

$$F^{sf} = \begin{cases} 0, & (h \geq \delta) \\ -F_0\left(1 - \frac{h}{\delta}\right) = \frac{F_0}{\delta}h - F_0, & (h < \delta) \end{cases} \quad (2.30)$$

where $h = r_{ij} - 2R$. When the value of h is smaller than some specified value δ, $\boldsymbol{F}^{sf}$ starts to act as a repulsive force. Its component can be obtained from geometry, that is,

$$\begin{aligned} F_x^{sf} &= F^{sf}\frac{\Delta x}{r_{ij}} \\ F_y^{sf} &= F^{sf}\frac{\Delta y}{r_{ij}}. \end{aligned} \quad (2.31)$$

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

3. Details of Program

Understanding and modifying code written by others are not easy tasks. In order to help understand, the structures and functions of this program are explained in this section. Because detailed explanation on writing serial LBM code can be found in plenty of source, here I focus on the features related MPI programming and particle model. This code is rudimentary and there exist lots of possible improvements and optimizations. Some of the possible optimization are pointed out in this section.

This code is written in C++, but its programming style is closer to C. Background on the advanced programing technique in C++ such as class or templet is not required to understand and use this code. It is parallelized with MPI library to be used both in shared-memory system and distributed memory system. Laminar incompressible flow, single component multiphase flow, multicomponent flow, nondeformable solid particle in fluid have been implemented. Following materials assume reader are familiar with C, C++, and MPI library and LBM programming.

The first header file named `input.h` start with setting precision.

```
// Precision
typedef double real;
#define MPI_REALNUM MPI_DOUBLE
//typedef float real;
//#define MPI_REALNUM MPI_FLOAT
```

Note that you should properly define the MPI datatype as well as C datatype.

3.1 Velocity set

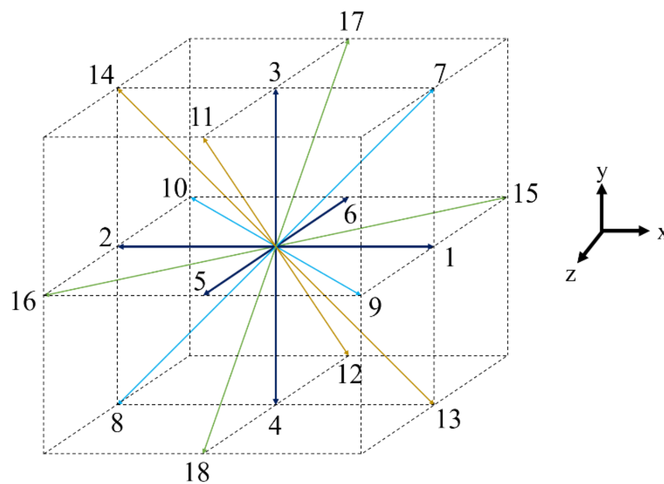


Figure 8. D3Q19 velocity set.

There still is not a general rule to numbering lattice velocity vectors. The numbering used in this program is shown in fig.00. Components of velocity set and corresponding weight coefficients are defined in the `headers/D3Q19.h`.

```
// velocity set
const int c[D][Q] =
   { {0, 1,-1, 0, 0, 0, 0, 1,-1, 1,-1, 0, 0, 1,-1, 1,-1, 0, 0},
     {0, 0, 0, 1,-1, 0, 0, 1,-1, 0, 0, 1,-1,-1, 1, 0, 0, 1,-1},
     {0, 0, 0, 0, 0, 1,-1, 0, 0, 1,-1, 1,-1, 0, 0,-1, 1,-1, 1} };

// weights
const real w1 = 1.0/3.0;
const real w2 = 1.0/18.0;
const real w3 = 1.0/36.0;
const real wt[Q] = {w1,w2,w2,w2,w2,w2,w2,w3,w3,w3,w3,w3,w3,w3,w3,w3,w3,w3,w3};
```

3.2 Domain decomposition

Domain can be adjusted in `Input/input.h` file by defining the number of lattice nodes. The total number of processors and the number of processors along each coordinate should be specified before running the program. Using `malloc()` in C or `new` in C++ with `MPI_Dims_create()` functions, it is possible to write a program that can run with any number of processors, however, setting number of processors in compile time reduce a lot of programing time. The size of arrays in each processor is than determined using domain size `I,J,` and `K` with the number of processors `P`. The actual size of local array is `[Ip+2][Jp+2]` including boundaries. Actual boundaries of whole domain are denoted with darker color in Fig.0.

```
// Domain size
const int I = 8;
const int J = 8;
const int K = 8;

// Number of processors
const int P = 4;
const int P_i = 2;   // # of processors along i
const int P_j = 2;   // along j
const int P_k = 1;   // along k

// Local array size
// - Local array size must be larger than
// - a particle's radius
const int Ip = I/Pi;
const int Jp = J/Pj;
const int Kp = K/Pk;
```

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

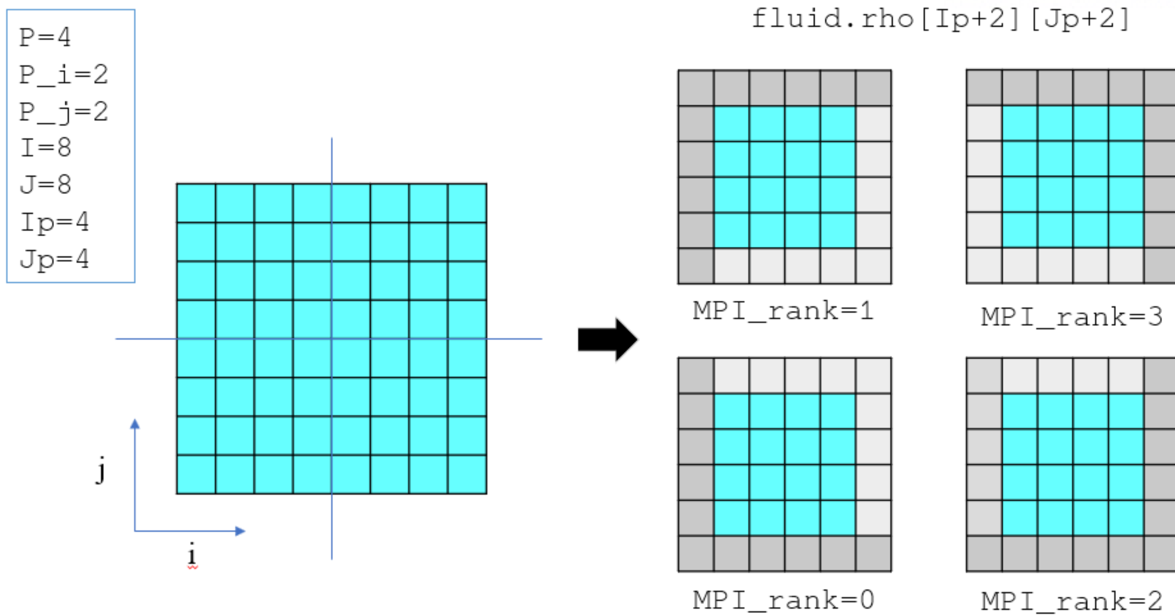


Figure 9. Domain decomposition.

3.3 Defining `FLUID` datatype

Structure is used to define fluid datatype as `struct FLUID` as shown in `headers/D3Q19.h`. This structure consists of relaxation time, macroscopic properties of fluid and its distribution functions. The concept of mass of a single particle is used in a number of literature on LBM, but the mass is simply set to one for simplicity in this program. `Ueq` is equilibrium velocity used in calculation for the equilibrium distribution function. Note that `f_afterCollision` can be omitted to save memory space if a more elaborated streaming process is implemented.

```
// arrays of fluid properties
struct FLUID
{
   real mass;  // mass of a single particle
   real tau;   // relaxation time

   // macroscopic properties
   real rho[Ip+2][Jp+2][Kp+2];
   real U[Ip+2][Jp+2][Kp+2][D];
   real Ueq[Ip+2][Jp+2][Kp+2][D];

   // distribution function
   real f[Ip+2][Jp+2][Kp+2][Q];
   real f_afterCollision[Ip+2][Jp+2][Kp+2][Q];
};
```

3.4 Finding neighboring processors

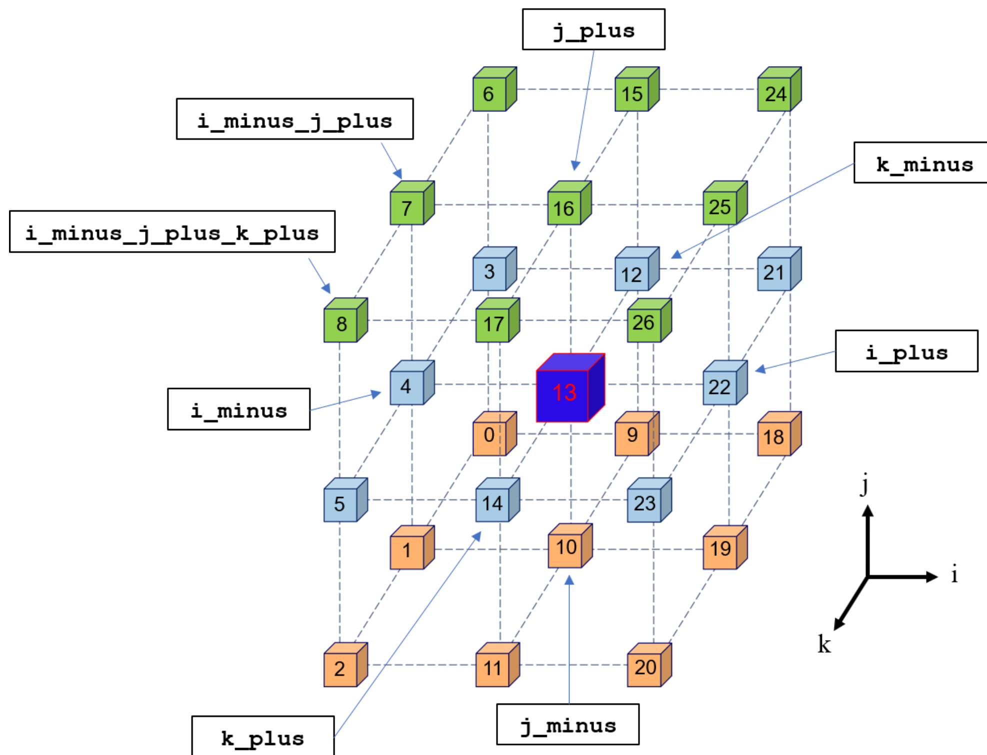


Figure 10. Processor grid. Neigh

With `MPI_Cart_shift()`, you can find the six neighboring processors along each coordinate. Neighbors along i direction are denoted by `i_plus` and `i_minus` as shown in fig.00. Neighbor along other axis are named in the similar way. Still, D3Q19 velocity set requires to identify the neighbors along the diagonal direction. For the particle simulation, every 26 neighboring processors need to be known to appropriately deal with particles which extend over more than two processors. Although it can be calculated using the number of processors along each axis, it becomes more complex when it comes to the processors at the corner with different boundary conditions. To deal with this issue, an array of `p_grid[P][6]` is introduced, whose first dimension implies the processor rank and the second dimension corresponds to the 6 neighboring processors which can be found with `MPI_Cart_shift()`. Additionally, it is required to name the neighboring processors properly and consistently. Consider a processor whose rank is set to 13 as shown in fig.00. For it, the processor 8 is named `i_minus_j_plus_k_plus` because to get the processor 8, it needs to move first in the negative direction, then positive j direction, finally positive k direction. The names for the other processors can be found in the first few lines of `src/particle.cpp`. To explain how `p_grid[P][6]` array is used, suppose again that we are on the processor whose rank is 13, and we

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

want to find who my `i_minus_j_plus_k_plus` neighbor is. First, its `i_minus` can be found using `p_grid[13][i_minus]=4`. Next, the processor 4's j_minus is `p_grid[4][j_plus]=7`. Finally, the processors 7's k_plus is `p_grid[7][k_plus]=8`. In short, `p_grid[p_grid[p_grid[13][i_minus]][j_plus]][k_plus]=8`.

	i_minus	i_plus	j_minus	j_plus	k_minus	k_plus
p_grid[13] =	4	22	10	16	26	14
p_grid[4] =	22	13	1	7	3	5
p_grid[7] =	25	16	4	1	6	8

Figure 11. Values in the `p_grid[P][6]` array.

3.5 Sequence of substeps

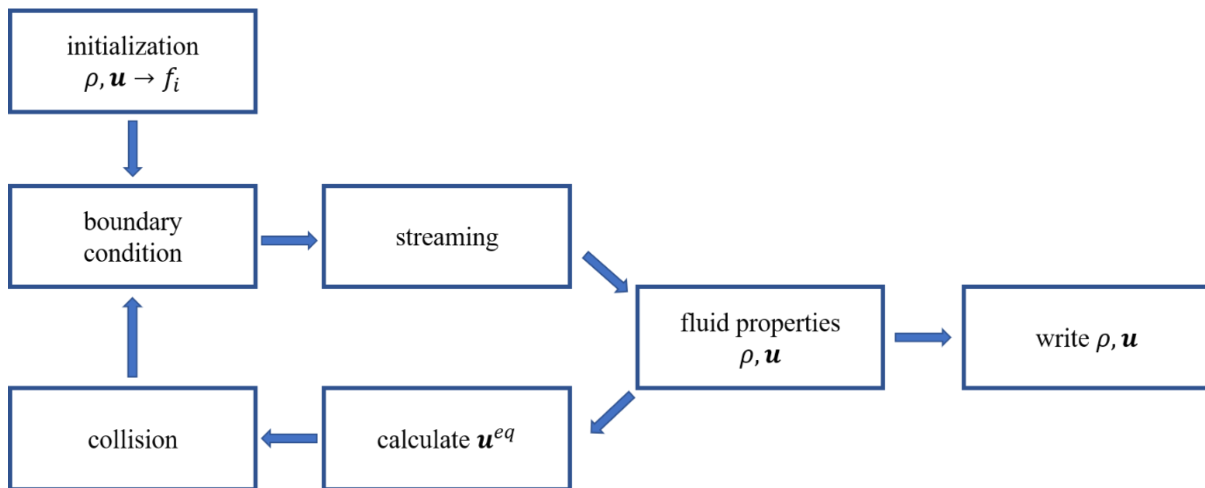


Figure 12. Sequence of substeps.

After initialized, the main loop might start with collision or streaming, or applying boundary condition. Even though it doesn't affect to the result, one should be aware of that when writing code for initialization or boundary condition. With the sequence of substeps shown in fig 00, initialization have to specify the distribution function as well as the fluid properties. Boundary condition should be applied to the boundary node outside of the computation domain and the proper distribution function need to be provided. If the loop starts with collision, initialization doesn't need to calculated distribution function, and if the boundary condition is applied after streaming process, extra boundary nodes are not

required, and the boundary condition will be applied to the missing distribution function of outermost node.

The main loop consists of five substeps. It starts by applying boundary condition. After that, streaming occurs. Programing technique for streaming and collision has already been dealt with by several literature[8, 17], so codes for them is not explained this paper. Using updated distribution function, the properties of fluid are calculated and saved depending on the values of `writingFrequency`. Next, equilibrium velocity is calculated. The effect the external force is included in the equilibrium velocity with SC forcing scheme. Collision occurs after that, then one cycle is completed.--

```
// Main loop
for (t=1; t<=totalTimeSteps; t++)
{
   // boundary condition
   periodicBoundaries();

   // streaming
   streaming(fluid);

   // get macroscopic fluid properties
   fluidProperties(fluid);

   // communication
   MPI_update_rho(comm, MPI_rank, p_grid, coords, fluid);

   // calculate equilibrium velocity wiht force
   equilibriumVelocity(fluid);

   // communication
   MPI_update_Ueq(comm, MPI_rank, p_grid, coords, fluid);

   // BGK collision
   collision_BGK(fluid);

   // write
   if (t % writingFrequency == 0)
   {
      if ( MPI_rank == 0 ) cout << "  Time: " << t << endl;
      write_density_local(MPI_rank, fluid, t);
      write_u_local(MPI_rank, fluid, t);
      write_v_local(MPI_rank, fluid, t);
      write_w_local(MPI_rank, fluid, t);
   }

}  // end of main loop
```

3.6 Initialization

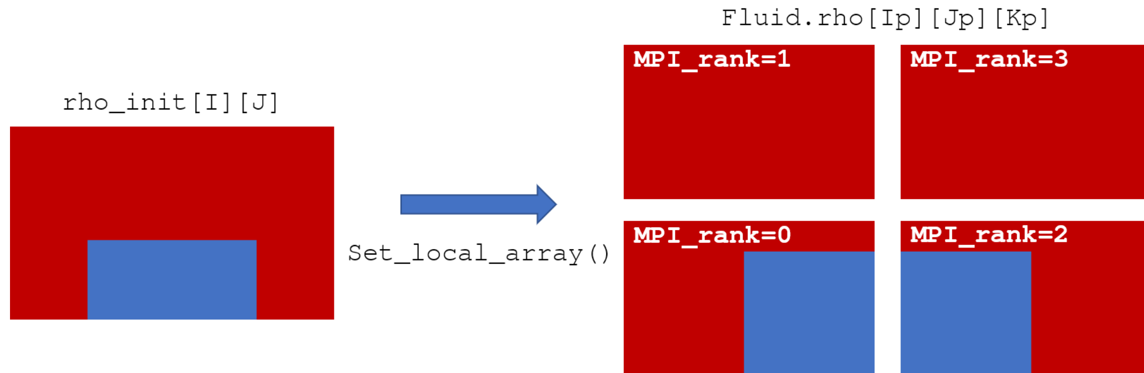


Figure 13. Initial condition decomposition.

Like any other method, LBM start with initialization. In this program, initialization consist of three sub-steps. First, a temporary array of full-size domain is created, and a user can specify the value of density and velocity on that array. Even though this temporary array helps to initialize fluid with a surface or droplet, it takes a huge amount of memory, so an improved way can replace this initialization method later. Next, each processor takes their portion of the domain from the temporary array and save them in their local array. Then the values in boundaries are transferred to the neighboring processors using MPI. Finally, an equilibrium distribution function is calculated from the values of density and velocity including the boundary node.

Fig00 describe how the local array is set from the full-size domain. `Set_local_array()` is in charge of this task.

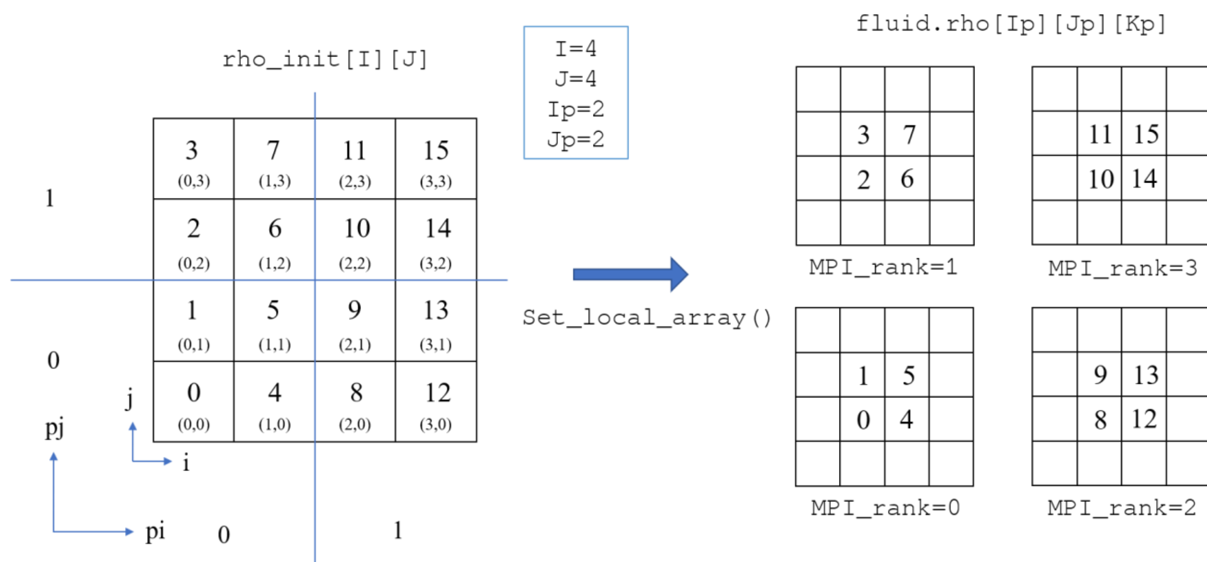


Figure 13. `rho_init[I][J]` and the local array.

```
void set_local_array(FLUID& fluid, real rho[][J][K], real U[][J][K][D], int* coords)
{
   int i,j,k,d;   // index for array of the full domain, "rho[I][J][K]"
   int ip,jp,kp;  // index for local array, "fluid.rho[Ip+2][Jp+2][Kp+2]"
   int pi,pj,pk;  // processor coordinates

   pi = coords[0]; pj = coords[1]; pk = coords[2];

   for ( ip=1, i=pi*Ip; i<pi*Ip+Ip; ip++, i++ )
   {
      for ( jp=1, j=pj*Jp; j<pj*Jp+Jp; jp++, j++ )
      {
         for ( kp=1, k=pk*Kp; k<pk*Kp+Kp; kp++, k++ )
         {
            fluid.rho[ip][jp][kp] = rho[i][j][k];
            for ( d=0; d<D; d++ ) fluid.U[ip][jp][kp][d] = U[i][j][k][d];
         }
      }
   }
}
```

Now the initial condition is split to each processor, but the value for the boundary node are still empty as shown in Fig. 14. The first communication occurs here using `MPI_update_fluid()`. After specifying density and velocity of the fluid, the distribution function needs to be determined. However, there are infinite number of possibilities because infinite number of microstate corresponding to one macrostate. The most common scheme is setting initial distribution function as equilibrium distribution.

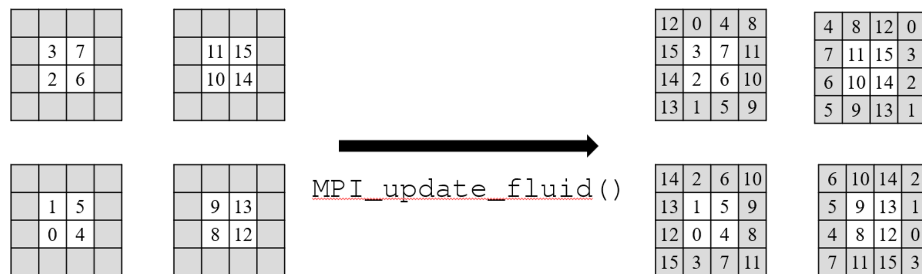


Figure 14. Local array. Boundary values are transferred to neighboring processors.

```
// initial condition
void MPI_initialization(int rank, MPI_Comm comm, int p_grid[][6],
                        int* coords, FLUID& fluid)
{
   if ( rank == 0 ) printf ("  Setting initial condition...\n");

   initialCondition_fluid(coords, rank);

   MPI_update_fluid(comm, rank, p_grid, coords, fluid);

   initialize_PDF_eq(fluid);  // equilibrium initial condition

   if ( rank == 0 ) printf ("  ...Done.\n\n");
}
```

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

3.7 Boundary conditions

There are two ways to apply boundary condition. First, boundary conditions can be applied directly to the missing distribution function at the outermost part of computation domain. For example, f_4, f_6, f_7 is missing for the upper boundary nodes as shown in fig00 (a). According to the applied boundary condition, proper value will be assigned to the missing distribution function. In this case, the boundary condition should be applied after streaming is finished. Alternatively, boundary nodes can be attached, and boundary condition is assigned in those node. In this program, boundary conditions are applied by specifying proper distribution function on those boundary nodes after collision process, before streaming occurs.

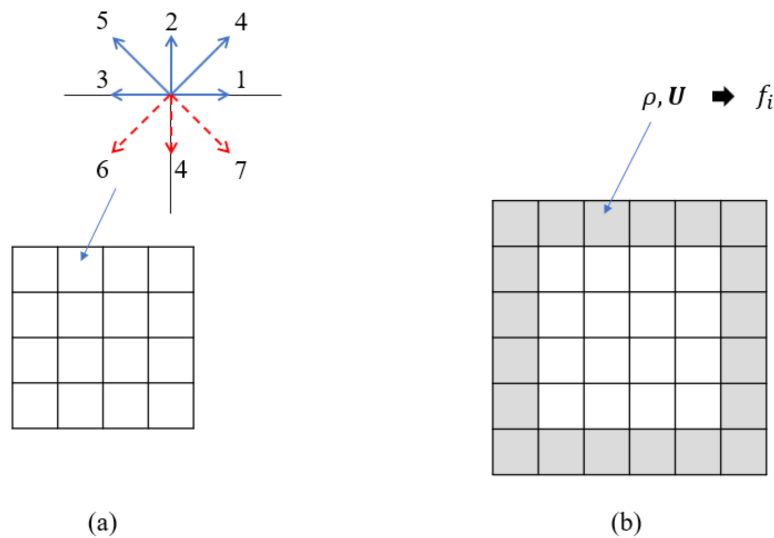


Figure 15. Two different scheme for applying boundary condition. (a) (b)

3.8 Communication

When writing LBM program with MPI, you need to choose the data that will be exchanged between processors. You can either pass the distribution functions or fluid properties. When the distribution functions are used, the communication occurs after the collision. Every point has 27 distribution functions, but only 5 of them are needed to be passed. However, packing and unpacking process for the distribution function will look rather tangled. Alternatively, the fluid properties can be exchanged. In this case, the communication occurs before collision because collision requires the updated fluid properties and the collision process in each node should include the boundary nodes. Each message carries four floating numbers; density of fluid and three components of fluid velocity.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

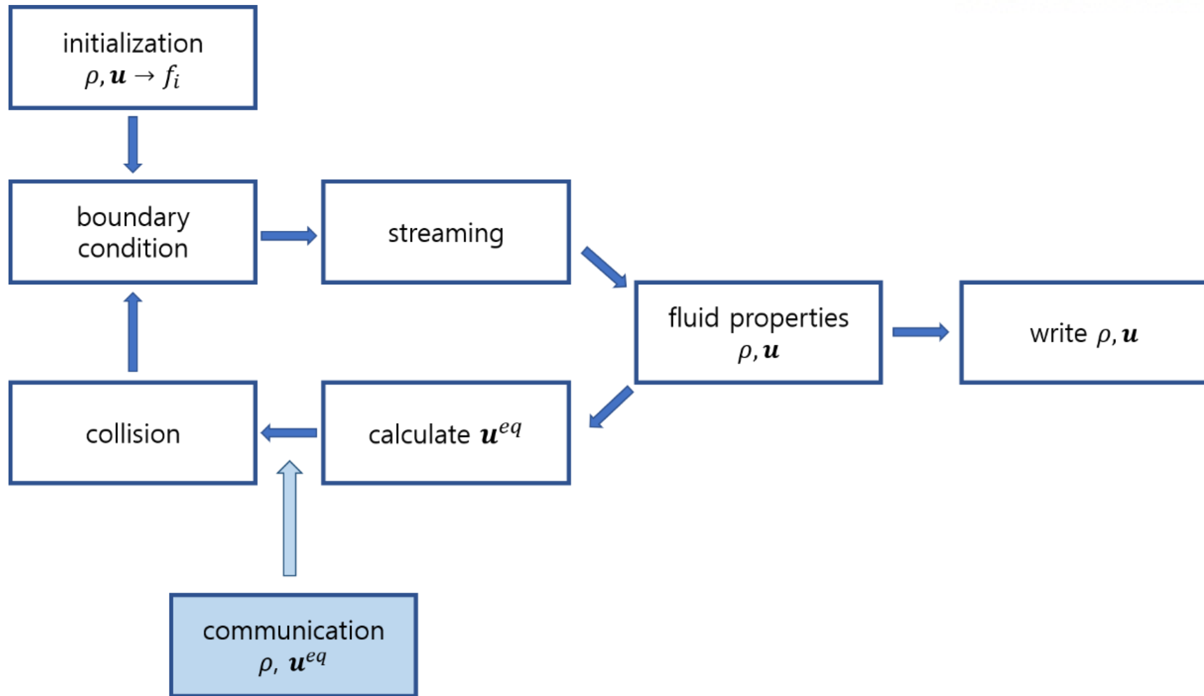


Figure 16. Sequence of substeps. Communication occurs before collision.

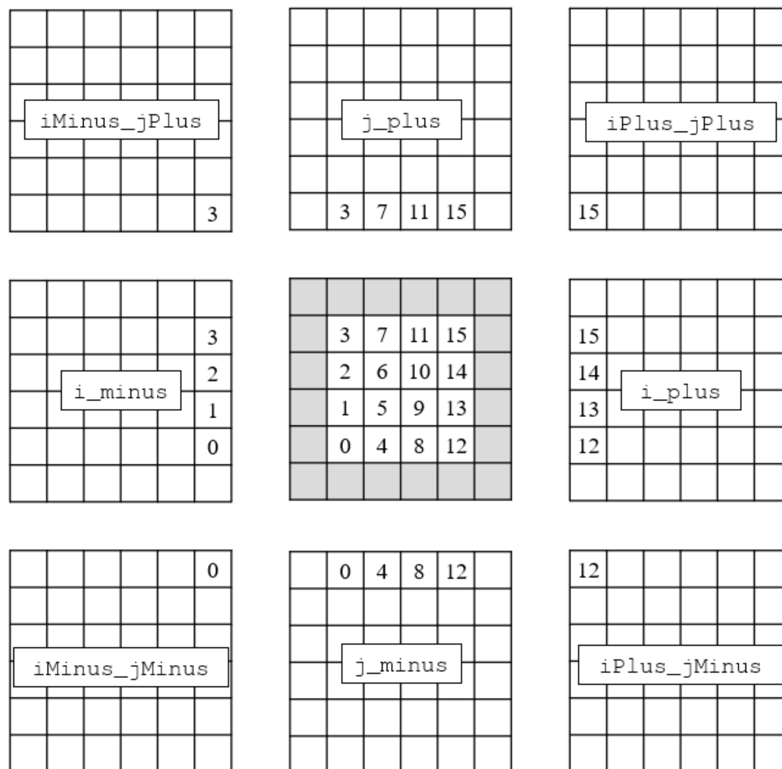


Figure 17. Local array after communication completed.

Communication process consist of three substeps: packing, passing and unpacking. Because the data to be passed on the boundary are not continuously saved in memory, we need to pack the data to be sent in an array and create an array to save the incoming message. After communication completed, unpacking process is required. The continuous data on the temporary array are written on the appropriate boundary node on the local array.

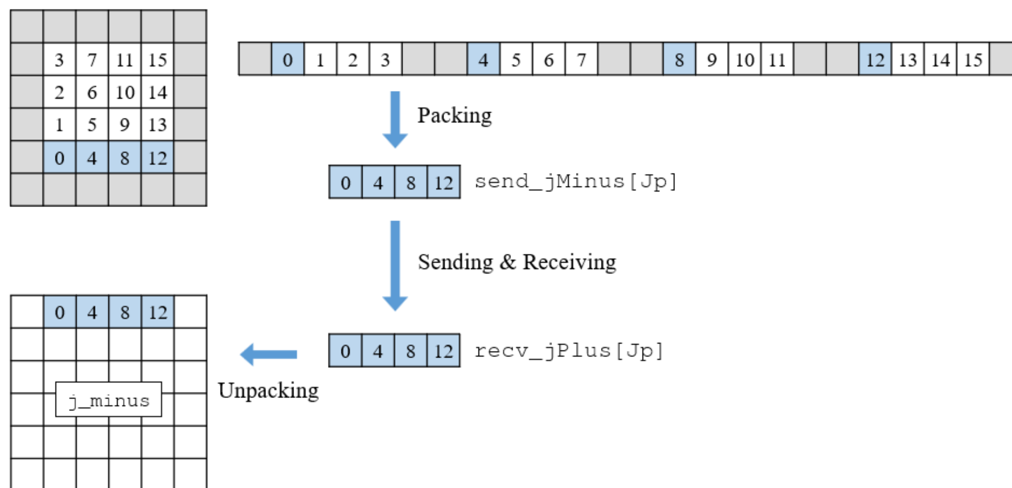


Figure 18. Substeps in communication.

```
// ---------------Packing------------------
// face
for ( j=0; j<Jp; j++ ) for ( k=0; k<Kp; k++ )
{
   send_iMinus[j][k] = fluid.rho[1][j+1][k+1];
   send_iPlus[j][k] = fluid.rho[Ip][j+1][k+1];
}
for ( i=0; i<Ip; i++ ) for ( k=0; k<Kp; k++ )
{
   send_jMinus[i][k] = fluid.rho[i+1][1][k+1];
   send_jPlus[i][k] = fluid.rho[i+1][Jp][k+1];
}
for ( i=0; i<Ip; i++ ) for ( j=0; j<Jp; j++ )
{
   send_kMinus[i][j] = fluid.rho[i+1][j+1][1];
   send_kPlus[i][j] = fluid.rho[i+1][j+1][Kp];
}
// -------------- Unpacking ----------------

// face
for ( j=0; j<Jp; j++ ) for ( k=0; k<Kp; k++ )
{
   fluid.rho[0][j+1][k+1] = recv_iMinus[j][k];
   fluid.rho[Ip+1][j+1][k+1] = recv_iPlus[j][k];
}
for ( i=0; i<Ip; i++ ) for ( k=0; k<Kp; k++ )
{
   fluid.rho[i+1][0][k+1] = recv_jMinus[i][k];
   fluid.rho[i+1][Jp+1][k+1] = recv_jPlus[i][k];
}
for ( i=0; i<Ip; i++ ) for ( j=0; j<Jp; j++ )
{
   fluid.rho[i+1][j+1][0] = recv_kMinus[i][j];
   fluid.rho[i+1][j+1][Kp+1] = recv_kPlus[i][j];
}
```

```
// ------------------------- Send & Receive---------------------------------
// face
MPI_Isend(send_iMinus, Jp*Kp, MPI_REALNUM, iMinus, tag, comm, &request[0]);
MPI_Isend(send_iPlus, Jp*Kp, MPI_REALNUM, iPlus, tag, comm, &request[1]);
MPI_Isend(send_jMinus, Ip*Kp, MPI_REALNUM, jMinus, tag, comm, &request[3]);
MPI_Isend(send_jPlus, Ip*Kp, MPI_REALNUM, jPlus, tag, comm, &request[2]);
MPI_Isend(send_kMinus, Ip*Jp, MPI_REALNUM, kMinus, tag, comm, &request[4]);
MPI_Isend(send_kPlus, Ip*Jp, MPI_REALNUM, kPlus, tag, comm, &request[5]);

MPI_Irecv(recv_iPlus, Jp*Kp, MPI_REALNUM, iPlus, tag, comm, &request[6]);
MPI_Irecv(recv_iMinus, Jp*Kp, MPI_REALNUM, iMinus, tag, comm, &request[7]);
MPI_Irecv(recv_jPlus, Ip*Kp, MPI_REALNUM, jPlus, tag, comm, &request[9]);
MPI_Irecv(recv_jMinus, Ip*Kp, MPI_REALNUM, jMinus, tag, comm, &request[8]);
MPI_Irecv(recv_kPlus, Ip*Jp, MPI_REALNUM, kPlus, tag, comm, &request[10]);
MPI_Irecv(recv_kMinus, Ip*Jp, MPI_REALNUM, kMinus, tag, comm, &request[11]);
```

3.9 Two component model

In order to calculated interaction force of SC model, the updated density is required. So, communication for density should be done before the interaction force is calculated. Since the value of interaction force is needed to calculate equilibrium velocity with the SC forcing scheme, communication for density should be carried out before calculating equilibrium velocity. Therefore, the resulting sequence of substeps called communication function twice. When the communication time is significant, it can cause the program to be slow down. Depending on the forcing scheme used, the sequence of substeps can be changed.

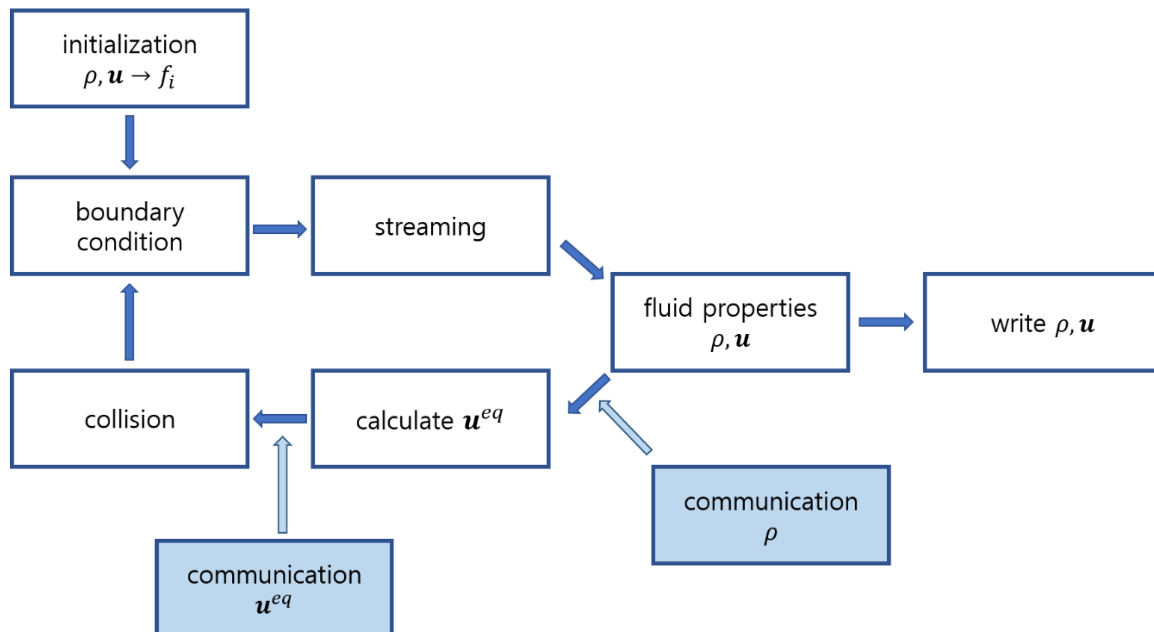


Figure 19. Sequence of substeps for the SC two component model.

3.10 Particle model

For the particle model, another layer of domain called `particle_domain` is added to the two layer of `fluidRED` and `fluidBLUE`. After the particle density is calculated for all the point on the particle domain, it acts like another SC fluid component, maintaining its shape.

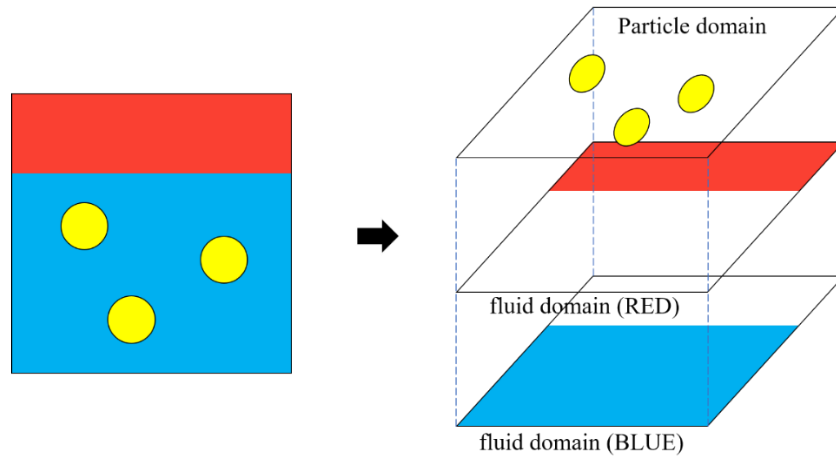


Figure 20. Three layers in the particle model.

```
FLUID fluidRED = {FLUID_MASS_RED, TAU_R, {0}, {0}, {0}, {0}};
FLUID fluidBLUE = {FLUID_MASS_BLUE, TAU_B, {0}, {0}, {0}, {0}};

PARTICLE_SET particle_set[Np] = {0};
PARTICLE_DOMAIN particle_domain = { {0}, {0} };
```

```
struct PARTICLE_DOMAIN
{
    real density[Ip+2][Jp+2][Kp+2];
    real U[Ip+2][Jp+2][Kp+2][D];
};
```

In order to set particle density on the `particle_domain`, the information in `PARTICLE_SET` structure is used. `PARTICLE_SET` contains the properties of particle (radius, interface width, mass and inertia) and position of the particle (processor to which it belongs, coordinate in the local array), velocity of the particle (translational velocity and angular velocity) and force on the particle (hydrodynamic force and torque due to fluid, body force and repulsive force from other particles).

3.10.1 Density

On the `particle_domain`, density field of particles is calculated with Eq 2.20. r is magnitude of a relative position vector **r** of a point on the particle with respect to the center of the particle.

```
struct PARTICLE_SET
{
   int processor;

   real radius;
   real interfaceWidth;
   real mass_single;

   real mass_total;
   real inertia;

   real G_RP;
   real G_BP;

   real X_center[D];
   real V_trans[D];
   real V_angular[D];

   real F_gravity[D];
   real F_hydro[D];
   real F_capil[D];
   real F_rep[D];
   real F_lub[D];
   real F_brown[D];
   real F_total[D];

   real T_hydro[D];
   real T_capil[D];
   real T_total[D];
};
```

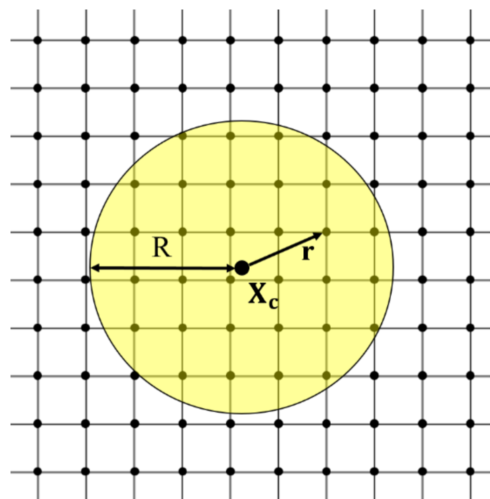


Figure 21. Particle domain. Nodes that are not occupied by the particle is set to zero.

```
void set_density_single(PARTICLE_SET p, PARTICLE_DOMAIN& p_d, int neighbor)
{
   double r;    // distance from the center
   int i,j,k,d;
   real X_center[D];

   // calculate particle's position
   calc_X_center(p, X_center, neighbor);

   for ( i=0; i<Ip+2; i++ ) for ( j=0; j<Jp+2; j++ ) for ( k=0; k<Kp+2; k++ )
   {
      // distance from the center
      r = ( real(i) - X_center[X] ) * ( real(i) - X_center[X] )
        + ( real(j) - X_center[Y] ) * ( real(j) - X_center[Y] )
        + ( real(k) - X_center[Z] ) * ( real(k) - X_center[Z] );
      r = sqrt(r);

      if ( r < (p.radius - 0.5*p.interfaceWidth) )
         p_d.density[i][j][k] += 1.;
      else if ( r > (p.radius + 0.5*p.interfaceWidth) )
         p_d.density[i][j][k] += 0.;
      else
         p_d.density[i][j][k] += 1.0 +
                               ( (p.radius - 0.5*p.interfaceWidth) - r )
                               / p.interfaceWidth;
   }
}
```

When more than two processors are used for a simulation, it is possible that a particle can extend over the neighboring processors, as described in Fig. 22.

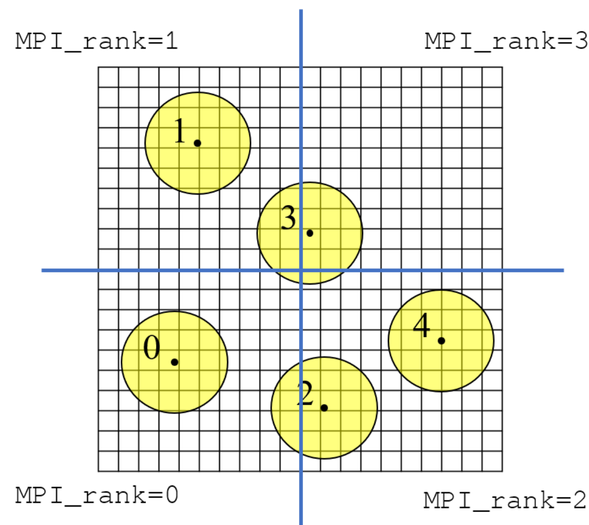


Figure 22. Particles on the whole domain.

For this task, `find_neighbor()` and `calc_X_center()` functions are implemented. First, `find_neighbor()` checks the particles position in the neighboring processors and to which

processor it belongs in order to determine if the particle will span over to its local domain. If so, it returns the processors relative position with respect to it using the naming rule described in Fig. 9. If the particle has no effect on it, it returns `-2`, and if the particle belongs to it, return `-1`. Using the return value of `find_neighbor()` and the particles position in the local domain of that processor, `calc_X_center()` adjust the position. For example, in the case of Fig. 23, `calc_X_center()` adjust the coordinate to (11,3). Then it can be used to set the particle domain in its domain in the ordinary way.

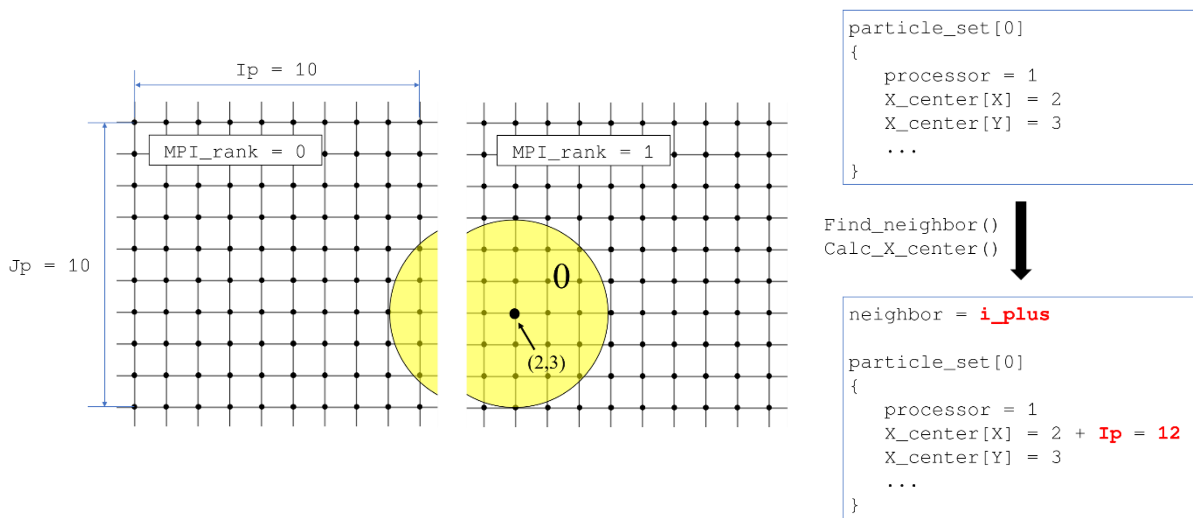


Figure 23. Particle which extends over the neighboring processors.

```
/* face */
else if (
            // if "iPlus" is not a wall
            !(i_BC1 != 1 && coors[0] == Pi-1)
            // if this particle belongs to "iPlus"
            && (p.processor == p_grid[rank][I_PLUS])
            // if this particle is close to the boundary
            && !(p.X_center[X] - p.radius - 0.5*p.interfaceWidth > 1.0)
        )
    neighbor = I_PLUS;    // 0
```

```
// neighbor: face
case I_PLUS    :  for ( d=0; d<D; d++ ) X_center[d] = p.X_center[d];
                  X_center[X] = p.X_center[X] + real(Ip);
                  break;
```

This process is repeated with every particle, and the result is summed up. When the particles position is like in Fig. 22, the resulting `particle_domain.density[Ip][Jp]` looks like Fig. 24.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

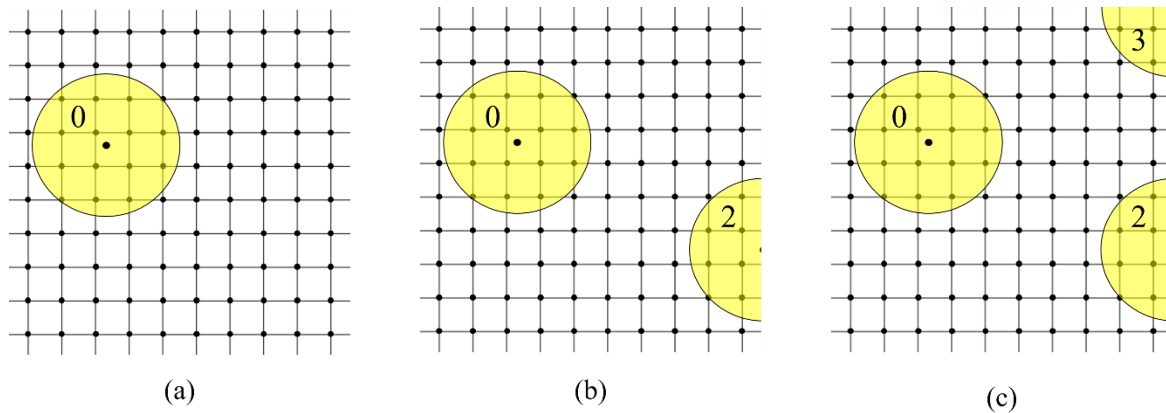


Figure 24. Particle domain. (a) When the loop index `np = 0`. (b) `np = 1`. Density of particle 2 is added. (c) `np = 3`. After the loop is finished, setting local particle density domain is completed.

```
// Span particle's density on the "particle_domain.density[Ip][Jp][Kp]"
// using "particle_set[Np]" ---------------------------------------------------
void set_density(const PARTICLE_SET p[], PARTICLE_DOMAIN& p_d,
                 int rank, const int* coords, const int p_grid[][6])
{
   int i,j,k,np;
   int neighbor;

   // reset density
   for ( i=0; i<Ip+2; i++ ) for ( j=0; j<Jp+2; j++ ) for ( k=0; k<Kp+2; k++ )
      p_d.density[i][j][k] = 0.0;

   for ( np=0; np<Np; np++ )
   {
      // "neighbor" is a relative position of a neighboring processor
      // that have a particle close to the boundary
      neighbor = find_neighbor(p[np], rank, coords, p_grid);

      if ( neighbor != -2 )   // if "particle[np]" is close to my domain
         set_density_single(p[np], p_d, neighbor);
   }
}
```

3.10.2 Local velocity

In the PSM, local velocity means the velocity of a point on a particle. For example, if the particle is rotating with respect to a stationary axis, the local velocity field looks like Fig. 25. Even though its translational velocity is zero, the local velocity could not be zero if the particle has an angular velocity. Fig. 26 describe how to incorporate the effect of angular velocity into the calculations of local velocity. The effect of angular velocity along other axis can be calculated in the similar way.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

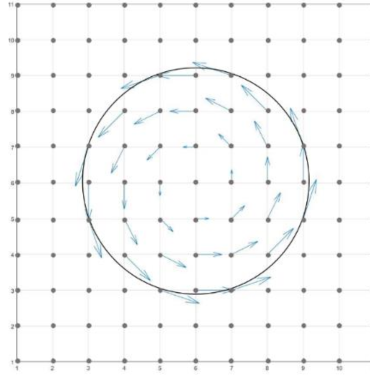


Figure 25. Particle velocity field when the particle is rotating with respect to a stationary axis.

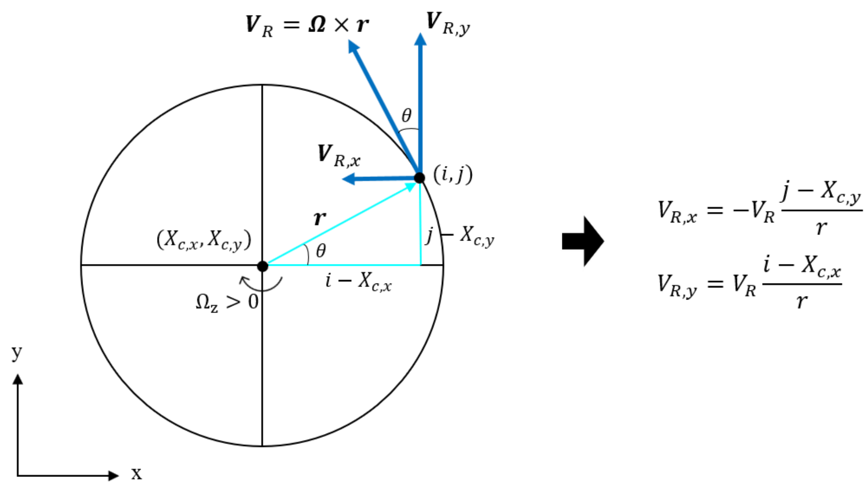


Figure 26. Local velocity of a particle.

3.10.3 Force gathering

Once local particle domain and velocity field are constructed on every processor properly, calculation of force become ordinary serial programing. However, when a particle extends over more than two processors, each processors calculate only part of the force, as described in Fig 27. Therefore, after force is calculated by `PARTICLE::calc_force_single()`, the partial forces distributed over the several processors have to be collected to the processor that is in charge of the particle using `PARTICLE::MPI_reduce_force()`. For example, to get an appropriate value of the force on the particle in Fig. 27, we need to summed up the forces and torque calculated at the processor 0 and processor 1.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

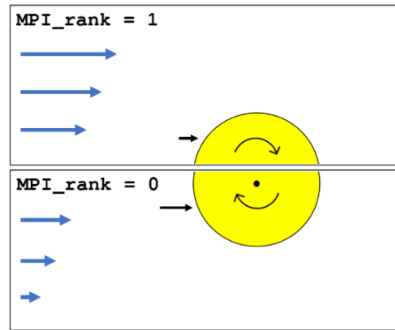


Figure 27. Fixed particle in shear flow.

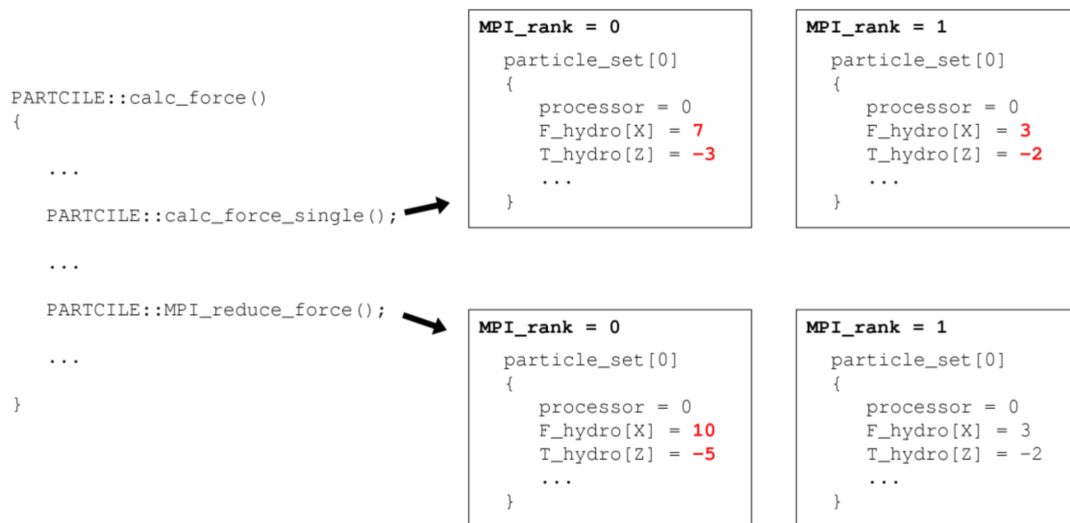


Figure 28. Values in the `particle_set[0]`

```
// gather forces from other processors
for ( np=0; np<Np; np++ )
{
   root = p[np].processor;

   // gather
   PARTICLE::MPI_reduce_force_particle(
               p[np],
               F_hydro_sum, F_capil_sum,
               T_hydro_sum, T_capil_sum,
               root, comm);

   // update force using "sum"
   if ( p[np].processor == myrank ) for ( d=0; d<D; d++ )
   {
      p[np].F_hydro[d] = F_hydro_sum[d];
      p[np].F_capil[d] = F_capil_sum[d];
      p[np].T_hydro[d] = T_hydro_sum[d];
      p[np].T_capil[d] = T_capil_sum[d];

      p[np].F_total[d] = p[np].F_gravity[d]
                       + p[np].F_hydro[d]
                       + p[np].F_capil[d]
                       + p[np].F_rep[d] + p[np].F_lub[d]
                       + p[np].F_brown[d];

      p[np].T_total[d] = p[np].T_hydro[d] + p[np].T_capil[d];
   }
}
```

```
// collect forces throughout processors
void MPI_reduce_force_particle(PARTICLE_SET p,
          real F_hydro_sum[], real F_capil_sum[],
          real T_hydro_sum[], real T_capil_sum[],
          int root, MPI_Comm comm)
{
   MPI_Reduce(p.F_hydro, F_hydro_sum, 3, MPI_REALNUM, MPI_SUM, root, comm);

   MPI_Reduce(p.F_capil, F_capil_sum, 3, MPI_REALNUM, MPI_SUM, root, comm);

   MPI_Reduce(p.T_hydro, T_hydro_sum, 3, MPI_REALNUM, MPI_SUM, root, comm);

   MPI_Reduce(p.T_capil, T_capil_sum, 3, MPI_REALNUM, MPI_SUM, root, comm);
}
```

3.10.4 Position update

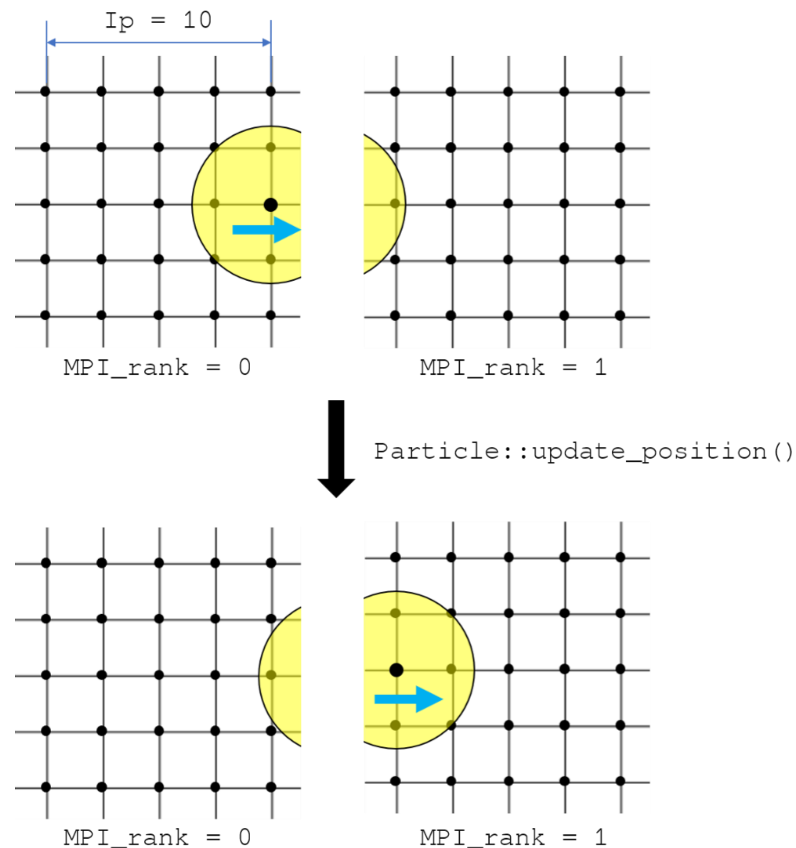


Figure 29. Particle moving across the boundary between two processors.

Particles position updated in every time step by the Eq 2.20. If the updated position is out of range of a local domain, that means the particle moves into a neighboring processor. After that, the neighboring processor become to be in charge of that particle, and the processor should know that the particle has

moved to its domain. This process is carried out with two functions, `Particle::update_position()` and `Particle::MPI_communication()`, as described in Fig. 30.

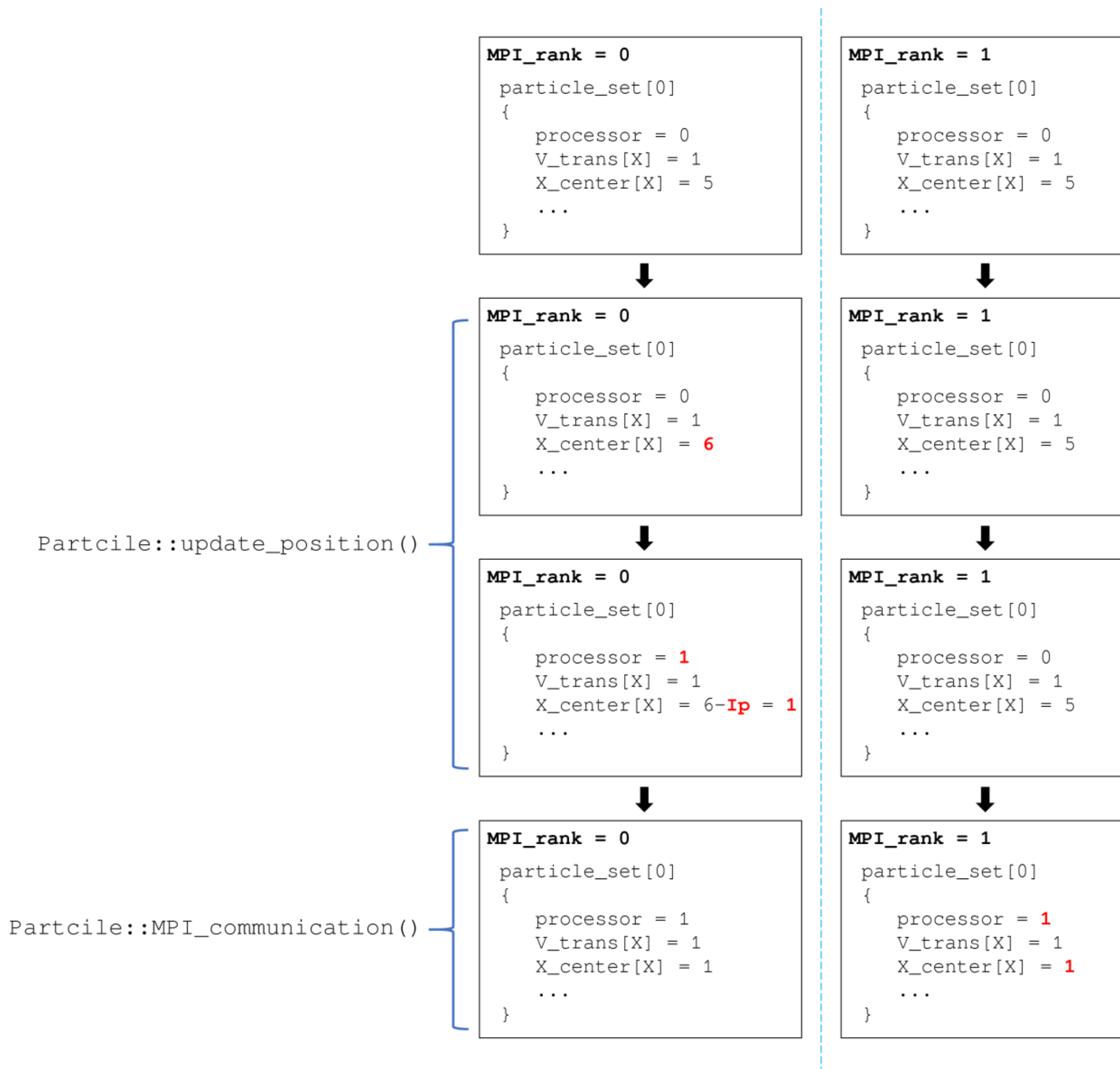
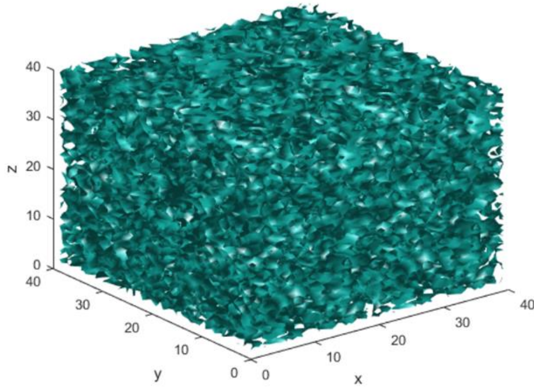


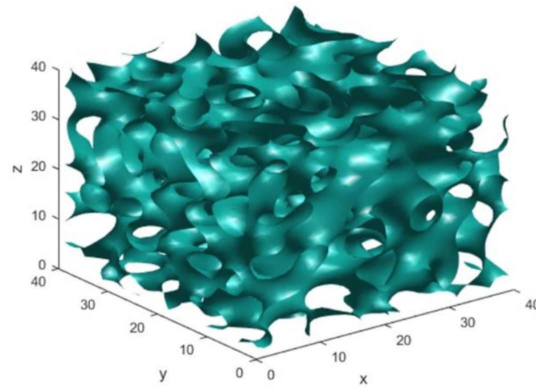
Figure 30. Values in `particle_set[0]` on each processor.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

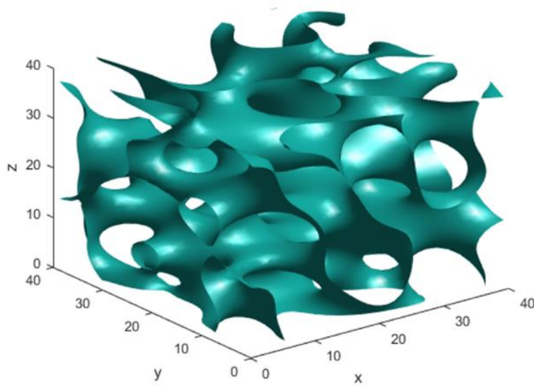
4. Examples



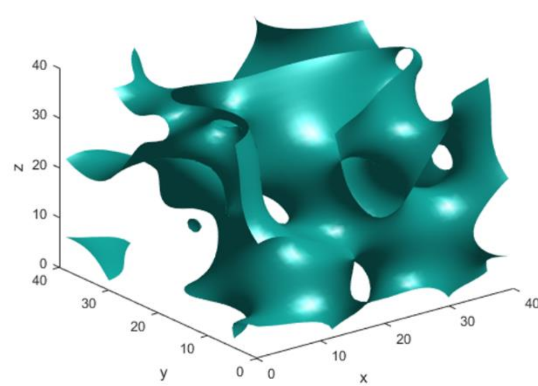
(a) t = 0



(b) t = 200



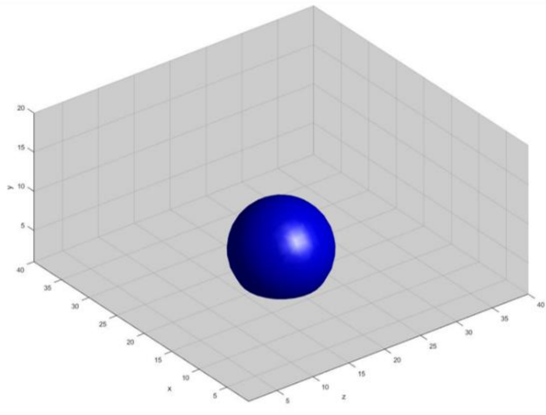
(c) t = 400



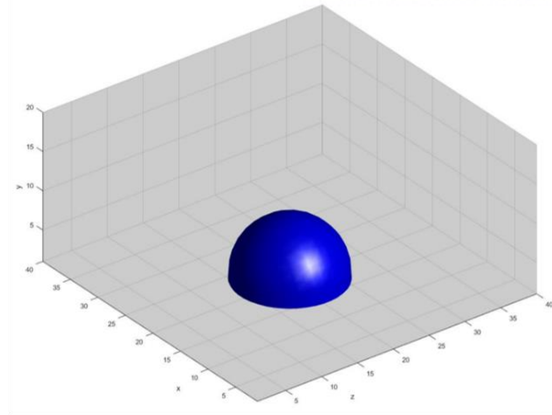
(d) t = 600

Fig 31. Simulation of 3D spinodal decomposition. Boundary of two component is shown here. They are mixed together at initial time, and separated as time goes.

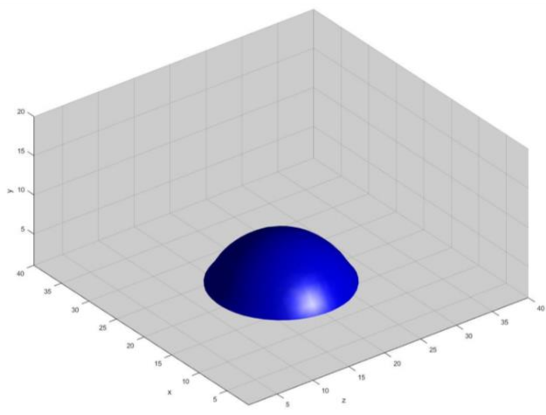
ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY



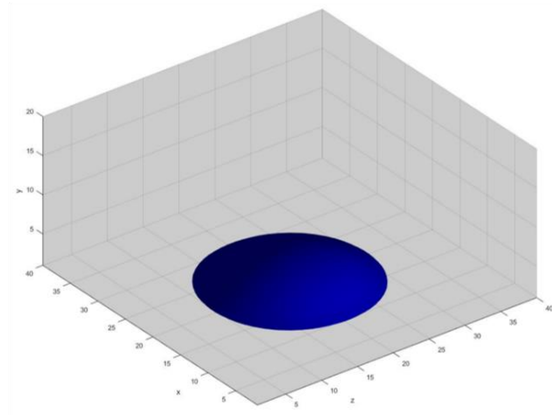
(a)



(b)



(c)



(d)

Figure 32. Isosurface of a density of a first fluid component at 0.5. Different contact angles can be simulated with SC model by adjusting the strength of interaction force between the fluid and the wall. Domain size is $40 \times 20 \times 40$. Initial condition is $12 \times 6 \times 12$ rectangle shape of the first fluid component surrounded by the other fluid component.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

5. Conclusions

The lattice Boltzmann methods is a relatively new method, but numerous models have been proposed to deal with the dynamics of particles in a viscous fluid that are consistent to the conventional methods. The pseudo-solid model is one of the recent lattice Boltzmann method for colloidal particles. Although it has not gained a lot of attention, its basic ideal is creative, and I think that idea have a potential to be applied to the other models. Therefore, it is important to study its own characteristics before use it to solve a physical problem. The method is installed in this code, so lots of numerical experiment and validation can be conducted with this code to improve the model. The code also considers the parallel computation, which is one of the main advantages compared with the conventional methods of the computational fluid dynamics. This code is carefully verified in the simple examples to avoid the unphysical behaviors of particle motions and thus can be further extended to study more complex problems. This code is not optimized and possibly has several mistake. I hope this paper will be used to understand the code for further development. Some possible application includes simulations of droplet, evaporation, capillary force and self-assembly.

References

[1] X. He, L.S. Luo, *Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation*, Phys. Rev. E **56**(6), 6811 (1997)

[2] X. Shan, X Yuan, H. Chen, *Kinetic theory representation of hydrodynamics: a way beyond the Navier–Stokes equation*. J. Fluid Mech. 550 (2006)

[3] U. Frisch, B. Hasslacher, Y. Pomeau, *Lattice-Gas Automata for the Navier-Stokes Equation*, Phys. Rev. Lett. **56**(14), 1505 (1986)

[4] G. R. McNamara, G. Zanett, *Use of the Boltzmann Equation to Simulate Lattice-Gas Automata*, Phys. Rev. Lett. **61**, 2332 (1988)

[5] F. J. Higuera, S. Succi, R. Benzi, Lattice Gas Dynamics with Enhanced Collisions, Europhys. Lett. **61**, 2332 (1988)

[6] S. Wolfram. *Statistical mechanics of cellular automata*, Rev. Mod. Phys. 55 (1983)

[7] D.A. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice Boltzmann Models* (Springer, New York, 2005)

[8] T. Kruger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, E.M Viggen, *The Lattice Boltzmann Method: Principles and Practice* (Springer, Switzerland, 2017)

[9] A.K. Gunstensen, D. H. Rothman, S. Zaleski, G. Zanetti, *Lattice Boltzmann model of immiscible fluids*, Phys. Rev. A **43**, 4320 (1991)

[10] X. Shan, H. Chen, *Lattice Boltzmann model for simulating flows with multiple phases and components*, Phys. Rev. E **47**, 1815 (1993)

[11] X. Shan, H. Chen, *Simulation of nonideal gases and liquid-gas phase transitions by the lattice Boltzmann equation*, Phys. Rev. E **49**, 2941 (1994)

[12] X. Shan, G. Doolen, *Multicomponent lattice-Boltzmann model with interparticle interaction*, J. Stat. Phys. **81**, 379 (1995)

[13] N. S. Martys, H. Chan, *Simulation of multicomponent fluids in complex three-dimensional geometries by the lattice Boltzmann method*, Phys. Rev. E **53**, 743 (1996)

[14] X. Shan, G. Doolen, *Diffusion in a multicomponent lattice Boltzmann equation model*, Phys. Rev. E **54**, 3614 (1996)

[15] M. R. Swift, W. R. Osborn, J. M. Yeomans, *Lattice Boltzmann Simulation of Nonideal Fluids*, Phys. Rev. Lett. **75**, 830 (1995)

[16] M. R. Swift, E. Orlandini, W. R. Osborn, J. M. Yeomans, *Lattice Boltzmann simulations of liquid-gas and binary fluid systems*, Phys. Rev. E **54**, 5041 (1996)

[17] H. Huang, M. Sukop, X. Lu, *Multiphase Lattice Boltzmann Methods: Theory and Application*, (John Wiley & Sons, Oxford, 2015)

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

[18] A. J. C. Ladd, *Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation. Part 1. Theoretical foundation*, J. Fluid Mech., **271** (1994)

[19] A. J. C. Ladd, *Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 2. Numerical results*, J. Fluid Mech., **271** (1994)

[20] A. J. C. Ladd, *Sedimentation of homogeneous suspensions of non-Brownian spheres*, Phys. Fluids **9** (1997)

[21] A. J. C. Ladd and R. Verberg, *Lattice-Boltzmann simulations of particle-fluid suspensions*, J. Stat. Phys. **104**, 1191 (2001)

[22] N. Q. Nguyen, A. J. C. Ladd, *Lubrication corrections for lattice-Boltzmann simulations of particle suspensions,* Phys. Rev. E **66**, 046708 (2002)

[23] F. Jansen, J. Harting, *From bijels to Pickering emulsions: A lattice Boltzmann study*, Phys. Rev. E **83**, 046707 (2011)

[24] Q. Xie, J. Harting, *From Dot to Ring: The role of friction in the deposition pattern of a drying colloidal suspension droplet*, Langmuir **34**, 5303 (2018)

[25] C. Sun, L. L. Munn, *Lattice-Boltzmann simulation of blood flow in digitized vessel networks*, Comput. Math. Appl. **55**, 1594 (2008)

[26] H, Shinto, D. Komiyama, K. Higashitani, *Lattice Boltzmann study of capillary forces between cylindrical particles*, Adv. Powder Technol. **18** (6), 643. (2007)

[27] G. Y. Liang, Z. Zeng, Y. Chen, J. Onishi, H. Ohashi, and S. Y. Chen, *Simulation of self-assemblies of colloidal particles on the substrate using a lattice Boltzmann pseudo-solid model*, J. Comput. Phys. **248** 323 (2013)

[28] M. Herlihy, N. Shavit, *The Art of Multiprocessor Programming* (Morgan Kaufmann, Boston, 2008)

[29] P. Pacheco, *An Introduction to Parallel Programming* (Elsevier, New York, 2011)

[30] M.C. Sukop, D.T. Thorne, *Lattice Boltzmann Modeling: An Introduction for Geoscientists and Engineers*. (Springer, New York, 2006)

[31] P. L. Bhatnagar, E. P. Gross, M. Krook, *A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems*, Phys. Rev. 94, 511 (1954)